Eemeli Miettinen

Designing a Secure Account Management System

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

8 December 2014

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Eemeli Miettinen Turvallisen tilinhallintajärjestelmän toteuttaminen 37 sivua + 2 liitettä 8.12.2014
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja	Lehtori Peter Hjort

Kiuru MSSP on Methics Oy:n tuote mobiilivarmennepalveluiden tarjoamiseen. Se on web-palvelinpohjainen ohjelmisto, joka perustuu European Telecommunication Standards Instituten (ETSI) standardeihin.

Tämän projektin tarkoitus oli korvata ja laajentaa Kiuru MSSP:ssä jo olemassa olevia tilinhallintasovelluksia turvallisilla sovellusratkaisuilla. Raportissa kuvataan järjestelmän arkkitehtuuri, sen toteuttamiseen liittyviä komponentteja ja käytettyjä teknologioita.

Työssä yhdistettiin olemassaolevat MReg (Mobile Signature Service Registration) -rajapinta ja REST API -palvelin uudella komponentilla, nimeltään Admin API. Admin API sisältää roolipohjaisen käyttöoikeuksien hallinnan ja käyttötapausten toteutuksen, jolla voidaan yhdistää olemassa olevia MReg-operaatioita.

Työssä käytetyt teknologiat osottautuivat erittäin tehokkaiksi. Erityisesti REST-pohjainen rajapinta helpottaa palvelun hallinnan integrointia aiemmin käytettyihin teknologioihin verrattuna.

Tuotetun järjestelmän avulla onnistuttiin yksinkertaistamaan sekä verkkopohjaisten että komentorivipohjaisten työkalujen tuottamista. Käyttäjien valtuuttaminen eri operaatioille tuli yhdenmukaiseksi ja se toteutettiin yleisesti hyväksytyillä ohjelmistoratkaisuilla. Tuotettu web-käyttöliittymä toimii esimerkkinä ja mahdollisena alustana web-palveluiden jatkokehityksessä.

Ratkaisun arkkitehtuurissa käytettiin yleisiä vastaavissa sovellusympäristössä käytettyjä tietoturvaratkaisuja. Lopullinen toteutus koostuu modulaarisista komponenteista ja niiden välisistä hyvin määritellyistä rajapinnoista.

Avainsanat	mobiilivarmenne, tilinhallinta, REST

Author(s) Title Number of Pages Date	Eemeli Miettinen Designing a secure account management system 37 pages + 2 appendices 8 December 2014
Degree	Bachelor of Engineering
Degree Program	Information Technology
Specialisation option	Software Engineering
Instructor	Peter Hjort, Senior Lecturer

Kiuru MSSP is a Mobile Signature Service Provider server product developed by Methics Oy. It is based on the European Telecommunication Standards Institute (ETSI) Mobile Signature standards.

The objective of the project described in this bachelor's thesis was to extend and replace existing service management systems with secure software solutions. This bachelor's thesis focuses on describing the design and implementation of the system. The used technologies are also described in detail.

Methics already had an existing protocol used for service management called MReg (Mobile Signature Service Registration). The system uses this protocol and extends it with a new product called Admin API. The Admin API provides role based account management and the ability to chain multiple MReg operations into use cases.

The technologies used in the project were much more powerful than the technologies Methics had before. Especially the REST-based API made application integration easy.

The finished product simplifies the development of web and CLI based applications. The produced web-interface works as an example and as a possible platform for further web-development.

The solution architecture exploits well-known security solutions that are used in similar software environments. The finished product consists of modular components and interfaces between them.

Keywords	mobile signature service, account management, REST

Table of contents

Abbrevations

AE	Acquiring Entity MSSP. An MSSP that routes requests received from Application Providers.
AP	Application Provider. An organization or a person providing a service that requires authentication.
API	Application Programming Interface. An interface that provides a set of tools provided for building software.
HMSSP	Home MSSP. An MSSP dealing with the current end user and transaction.
HTML	HyperText Markup Language. A markup language for creating web pages.
HTTP	HyperText Transfer Protocol. An application protocol to transfer hypertext.
ICCID	Integrated Cirtuit Card Identifier. A unique identifier number stored in a SIM card.
IMSI	International Mobile Subscriber Identity. A number used to uniquely identify a user of a cellular network.
JSON	JavaScript Object Notation. A format for representing objects composed of key-value pairs.
LoA	Level of Assurance. Level of confidence for the authentication method.
MReg	MSS Registration. Methics' extension to the ETSI TS 102 204 MSS_Registration messages.
MSISDN	Mobile Station International Subscriber Directory Number. A number for identifying a subscription in a GSM or UMTS network. This is commonly known as a mobile phone number.

MSS	Mobile Signature Service. A service that enables the generation of Mobile Signatures.
MSSP	MSS Provider. An entity that provides Mobile Signature Service.
MUTK	Mobile User Toolkit. Command line software used to manage Kiuru MSSP accounts.
REST	Representational State Transfer. An architectural style for web service development.
SOAP	Simple Object Access Protocol. An XML based communication protocol for exchanging structured information.
SPML2	Service Provisioning Markup Language 2.0. An XML-based standard for service provisioning.
XML	Extensive Markup Language. Markup language for representing structural data.

1 Introduction

Methics is a Finnish company that provides software products for mobile digital signing. Methics' main product is the Kiuru MSSP. The Kiuru MSSP is an MSSP (Mobile Signature Service Provider) server that acts as a modular SOAP (Simple Object Access Protocol) message router. Routing the SOAP messages requires various accounts on the MSSP. This bachelor's thesis focuses on describing the design and implementation of a management system for these accounts. The project was done for Methics Oy. The next chapter covers Mobile Signature Service in order to explain what kind of accounts the system needs to handle.

1.1 Introduction to Mobile Signature Service

Mobile Signature is a digital signature on a SIM card (Subscriber Identity Module). It allows a mobile user to securely sign transactions with their mobile device. Signed transactions can be used for user authentication and approval of commercial transactions. [1.]

Mobile Signature Service (MSS) is a service for Application Providers (AP) for requesting digital signatures from mobile users. The two main principles of MSS are that the signature processing is secure and the AP does not need to know the mobile user's operator.

The Mobile Signature Service is based on a four corner model (see Figure 1). The four entities in the model are mobile user, Application Provider, Acquiring Entity MSSP (AE) and Home MSSP (HMSSP). [2.]

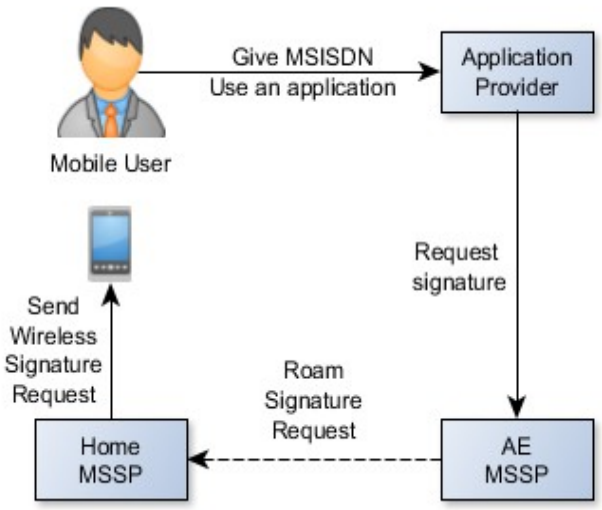

Figure 1: Four Corner Model [2.]

1.1.1 Application Provider

An Application Provider (AP) is any application that requires user authentication [3.]. Typically an AP is an HTTP server like a web banking service, Identity Provider (IDP) application for single-sign-on service or a Remote Authentication Dial In User Service (RADIUS) server for network access.

1.1.2 Acquiring Entity

The Acquiring Entity is an MSSP that routes requests received from Application Providers [4.]. In order for an AP to send signature requests without knowing the Mobile User's operator, it needs to connect to an Acquiring Entity (AE). AE verifies the APs with mutual TLS authentication.

The AE acts as an entry point in an MSSP mesh. The mesh is a network connecting all MSSPs. Typically the AE does not have a connection to all of the HMSSPs in the mesh. Instead it creates a plan for the messages to go through the mesh to the mesh end point HMSSP. [4.]

1.1.3 Home MSSP

The HMSSP authorizes requests received from the mesh. It identifies the message type and directs the requests to a wireless network. The HMSSP receives the mobile user's signature and returns the response by using the same route as the original request. [4.]

1.1.4 Real World Example

A typical use case for MSS is when a user wants to log in to a service that requires strong authentication. This kind of authentication is needed in, for example, bank services.

The user gives their MSISDN (Mobile Station International Subscriber Directory Number) to the Application Provider (see Figure 2). MSISDN is the same as the mobile user's mobile phone number. [2.]

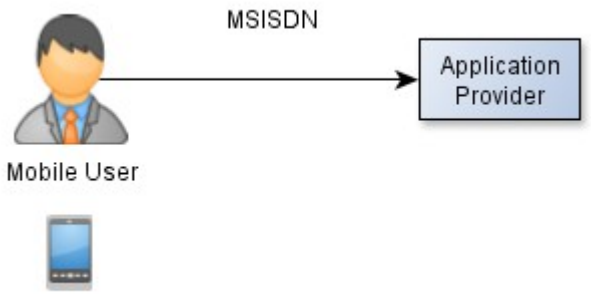

Figure 2: User Enters MSISDN [2.]

The AP creates a signature request containing the given MSISDN and sends it to an Acquiring Entity it has a contract with (see Figure 3). [2.]

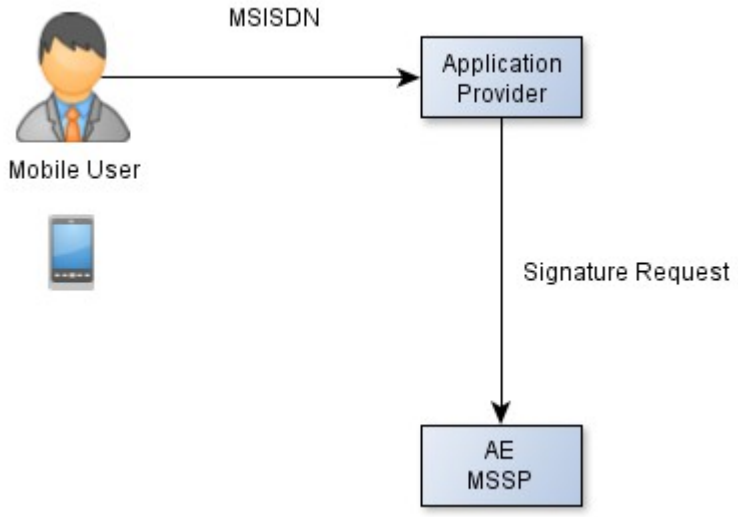

Figure 3: AP Sends Signature Request [2.]

The AE reads the MSISDN from the request and resolves the mobile user's HMSSP. The AE then routes the request through a mesh to the user's mobile operator's HMSSP (see Figure 4). [2.]

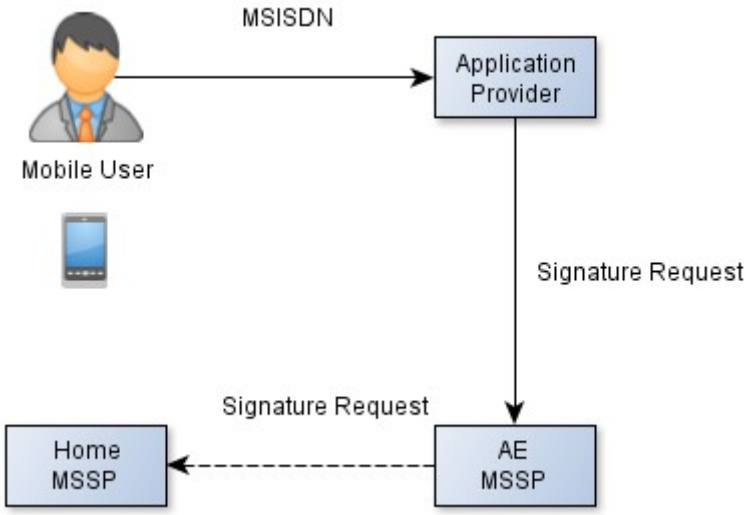

Figure 4: AE Routes Signature Request to HMSSP [2.]

The HMSSP receives the request and creates a wireless request that the SIM application understands. HMSSP sends the wireless request to the user's phone over the wireless network (see Figure 5). [2.]

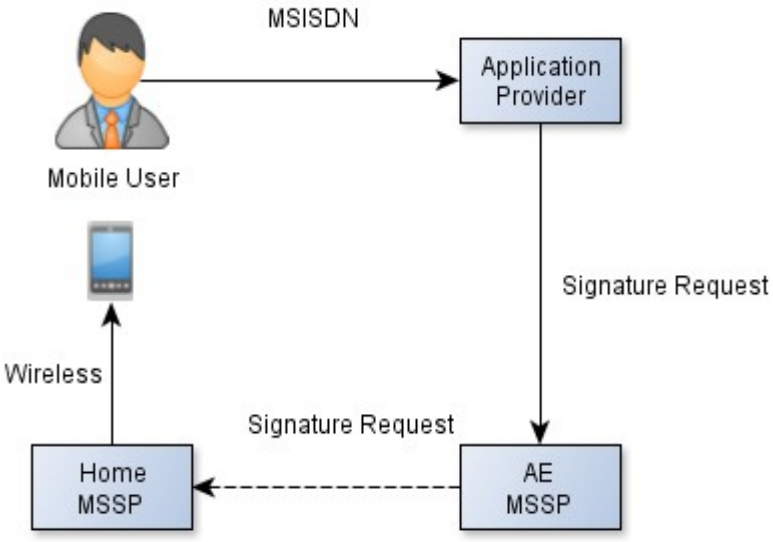

Figure 5: HMSSP Forwards Signature Request via Wireless Network [2.]

The user enters their PIN code to authenticate to the SIM application. The application responds to the HMSSP with a signature response. The HMSSP verifies the user's signature and certificate and responds to the AE using the same route as the original request. The AE sends the response to the AP who allows the user to access the service. [2.]

1.2 Introduction to Account Management

The MSSP system contains three major types of system entities: Mobile Users, APs and MSSPs. Each of the entities has a set of properties which are stored into an account specific to that entity. Typical properties are certificates and URI identifiers. Account management is the process of managing these properties. [5.]

1.2.1 Acquiring Entity

As shown in Figure 6, the AE holds MSSP and AP accounts. The AP accounts define Application Providers registered with the AE. They are used to determine whether an AP is allowed to send requests or not. The AP account can hold certificates required for TLS authentication. [5.]

The MSSP accounts define all known MSSPs. They are used by the AE to determine where it can route messages. [5.]

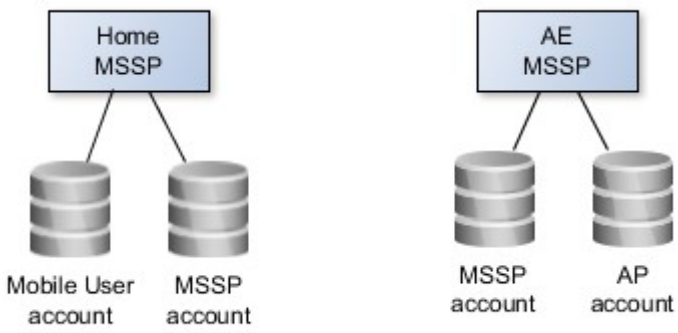

Figure 6: Entity Account Databases [5.]

1.2.2 Home MSSP

Home MSSP holds MSSP and Mobile User accounts. The Mobile User accounts define users with a Mobile ID subscription. They have properties that describe how to identify the user for the service. These properties can be, for example, MSISDN and IMSI (International Mobile Subscriber Identity). The IMSI is a number that uniquely identifies a subscription in a mobile network. [5.]

The MSSP accounts define all known MSSPs. They are used by the HMSSP to determine where it can route messages. [5.]

1.2.3 Account Management Roles

Account management is done by different people depending on the issue. Limiting account management by role allows to narrow the user's rights down to the minimum required by their position.

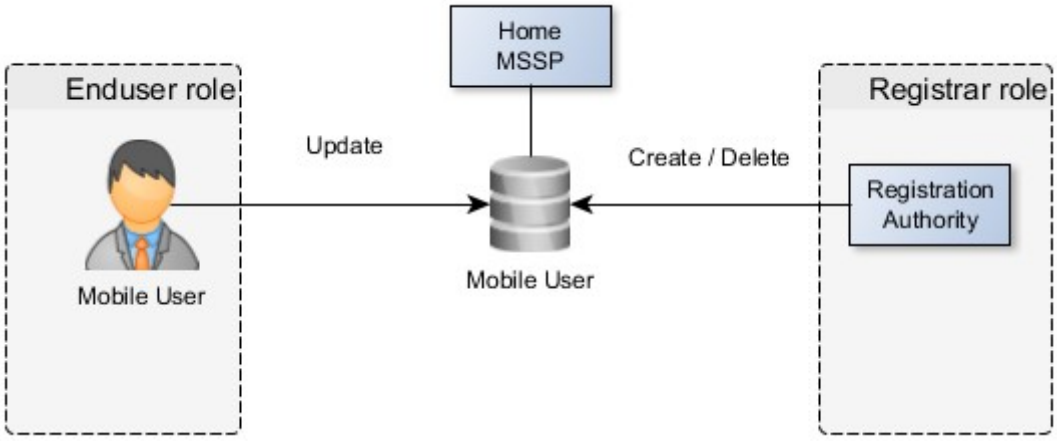

Figure 7: Examples of Management Roles [6.]

The example shown in Figure 7 contains two roles: end user and registrar. In the example the Mobile User is only allowed to update some of their account properties like their home address. Additionally a Registration Authority having the registrar role is able to register new Mobile User accounts and remove old ones.

2 Requirements

Methics has many separate entity management applications. At the start of this project each of the applications had its own security framework, user interface and installation procedure. Also, generic access control was missing. New web applications were required to be developed mostly from scratch.

The main objective of this project was to simplify the development process of new web applications. A starting requirement for the project was to improve the process by implementing an administration server as seen in Figure 8. In addition to the web implementation, the tools had to be command line interface (CLI) compatible.

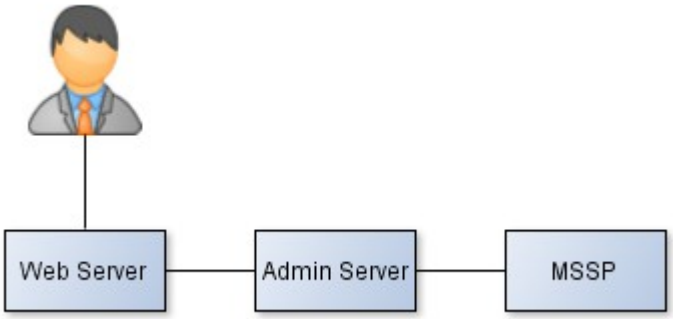

Figure 8: Kiuru Admin Overview [6.]

Another requirement was that the administration server must provide an interface for web application development. The interface should also provide access to MSSP account management operations.

2.1 Account Management Operations

Kiuru MSSP has over one hundred supported SOAP based management operations [5.]. Another requirement for the project was that the administration system must support these operations.

2.2 Authentication and Authorization

Since the application was to be used for MSSP account management, the authentication had to be secure. For this reason, the authentication was required to be based on well-known Java security mechanisms.

Application authorization had to be based on user roles. Moreover the project had to use standard Java principals. Authenticated users were presented as Java principals. Each principal could be assigned multiple roles and permissions could be defined for each role separately.

2.3 Compatibility

The application had to be compatible with existing Kiuru MSSP systems. This included being compatible with the existing W3C SOAP 1.2 frameworks, HTTP communication, TLS authentication, Java security framework and Java Servlet 3.0.

3 Analysis of Technologies

This chapter describes the major technologies and tools used in the project.

3.1 Development

The software described in this bachelor's thesis was programmed in Java 7. This is because Methics focuses on Java software development. Eclipse IDE (Integrated Development Environment) is used for Java source code editing and management.

Aptana Studio software was used for web development. Aptana Studio is an IDE for web applications. It was chosen because its look and feel are very close to Eclipse. It also provides code assist for HTML, CSS and JavaScript. [7.]

All project components were built and packaged with Apache Ant. Ant can be configured to compile Java code and copy all necessary libraries in place with an XML configuration file called build.xml. The project build files were integrated with the company's internal build system. [8.]

3.2 Platform

3.2.1 Servlet 3.0

Servlet 3.0 is a specification defined in a Java Specification Request made by Java Community Process. It defines the requirements for Java servlets and servlet containers. [9.]

Servlets are Java based web components that extend the capabilities of a server. They are used to handle requests and responses. The main functionality of the servlets is to generate dynamic content. [9.]

A servlet container is a part of a web server that handles and manages servlets and their life-cycle. The container can be an add-on component to a web server or a stand-alone host web server. Figure 9 shows a typical Servlet 3.0 environment. [9.]

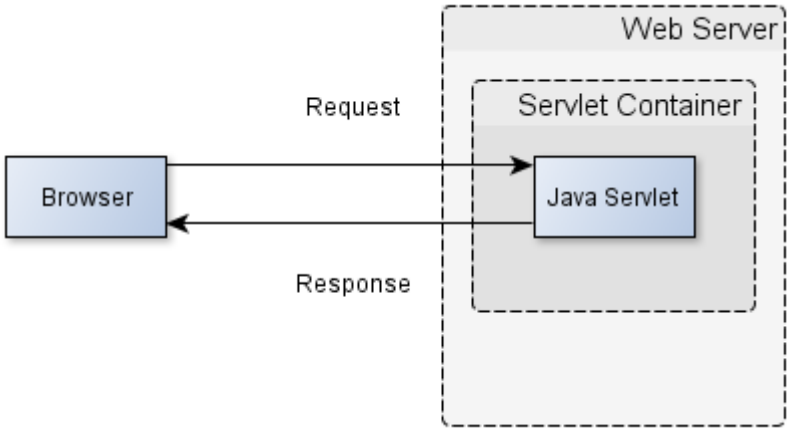

Figure 9: Servlet 3.0 Environment [9.]

3.2.2 XML Marshaller

An XML marshaller and unmarshaller is used for converting Java objects to XML and vice versa.

This project used the open source Castor XML marshaller. Castor was selected because in general it provides a good standard compliance and works well with complex XML schemas. The main uses for it were marshalling and unmarshalling SOAP messages. Castor was also used for reading service configuration. [10.]

3.3 Communication Environment

3.3.1 SOAP

SOAP is an XML based communication protocol for exchanging structured information. It provides a basic messaging framework for web services. [11.]

The protocol consists of three parts (see Figure 10). The first part is an envelope defining the message structure and processing. The second part is a set of encoding rules that define application specific data types. The third part is a convention for representing remote procedure calls (RPC). [11.]

A SOAP message is an XML document that consists of at least an envelope and a body. Optionally the document can also contain a header and a fault. The body, header and fault elements are always wrapped inside the envelope. [11.]

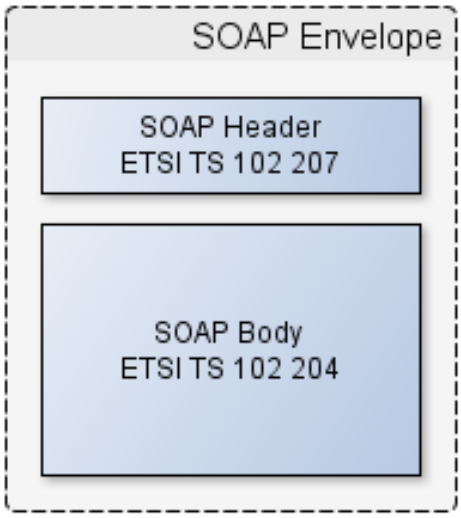

Figure 10: MSS SOAP Message Structure [11.]

As shown in Figure 10, the MSS SOAP message body is defined in the ETSI TS 102 204 specification. The SOAP header for MSS messages is defined in ETSI TS 102 207 specification.

ETSI TS 102 204 is a specification for a mobile signature web service interface. It specifies message types for signature requests, status requests, registration requests and receipt requests. It also defines three different roaming modes: synchronous, asynchronous client-server and asynchronous server-server. [12.]

ETSI TS 102 207 is a specification for roaming in mobile signature services. It defines how the message travels between MSSP entities in a mesh. This information is stored in the SOAP header. It is agnostic towards the way the message is transferred. The message transfer can be either synchronous or asynchronous. [4.]

3.3.2 Representational State Transfer

Representational State Transfer (REST) is an architectural style to develop web services. It allows a distributed system to have properties such as scalability and portability. As REST is an architectural style, it is not a formally defined protocol. [13.]

REST is stateless, meaning that no session is stored on the server side. The client holds the session. Each request contains all required information to service the request. The trade-off of the stateless constraint is that the network performance may decrease. [13.]

3.3.3 MReg

MSSP Registration is an extension to the ETSI TS 102 204 MSS_Registration messages. The ETSI standard defines extension points where this kind of extension is supported. The three main types of MReg operations are provisioning operations, certification operations and wireless operations. [15.]

Provisioning operations are used for account management. They can be used to create, read, update or delete accounts and account attributes. Certification and wireless operations exploit the accounts created by the provisioning operations.

Certification operations are used to access a Certificate Authority. They can request or revoke Mobile User or Entity certificates.

Wireless operations are used to communicate with a wireless network. They can be used to send commands to the SIM card. A wireless operation could be used, for example, to ping the SIM card to check if it has the required software installed.

3.3.4 MSSAPI

MSSAPI is a Java library developed by Methics. It is used for communicating with the MSSP. It provides a standard set of functions defined by ETSI TS 102 204 specification. [15.]

The MSSAPI provides a simple Java API for creating MSS SOAP messages. It uses Castor data types generated from an XML schema to build the messages. It supports MReg, SPML2 and SignatureRequest messages. Figure 11 shows the relation between a Java Application, the MSSAPI and the MSSP.

Figure 11: MSSAPI Overview

3.4 User Interface

3.4.1 jQuery

jQuery is a JavaScript library that simplifies client-side scripting. It provides simple syntax for many common scripting functionalities such as document navigation and event handling. jQuery also provides cross-browser compatibility. It handles all cross-browser inconsistencies and provides a unified interface. [16.]

3.4.2 Bootstrap

Bootstrap is an open source front-end framework. It consists of a collection of tools for creating web applications and websites. It contains HTML and CSS-based templates for web interface components. [17.]

Bootstrap also comes with a set of JavaScript components. They provide additional functionality for existing interface elements. The JavaScript components are in the form of jQuery plugins. [17.]

4 Solution Architecture

This chapter describes the architectural design of the project solution. The solution is based on a three tier architecture that contains a Content Server, Admin Server and an MSSP. (see Figure 12) The web content is hosted on the Content Server. The Content Server forwards the JSON (JavaScript Object Notation) requests to the Admin Server and terminates all other HTTP requests. The Admin Server is a gateway between the JSON requests and the MSS SOAP requests.

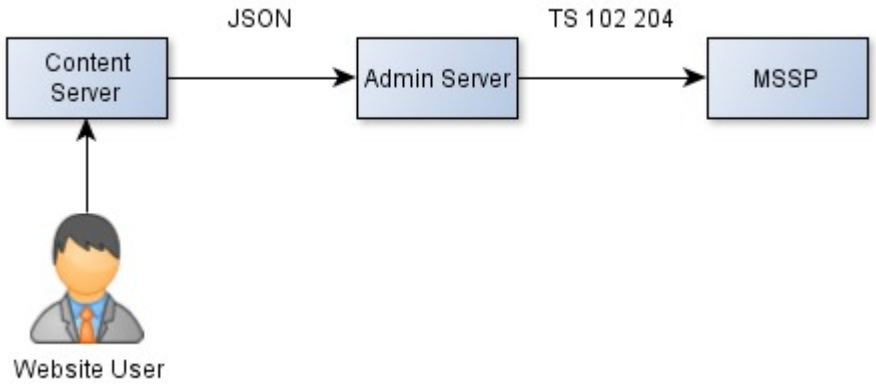

Figure 12: Solution Overview [18.]

4.1 Three-tier Architecture

The design of the administration system is based on a three-tier communication architecture. This means that the system is separated to presentation, logic and data tiers as seen in the following figure.

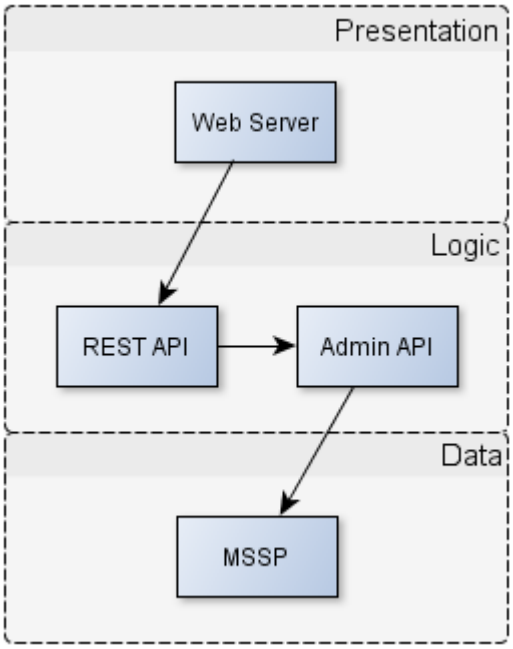

Figure 13: Three-tier Architecture

The presentation tier contains the user interface consisting entirely of HTML, CSS and JavaScript. The Content Server constructs the necessary JSON requests based on the user actions. The Content Server sends the JSON requests to the REST API residing on the logic tier.

The logic tier contains all of the administration business logic. It contains the user authentication and operation authorization systems. The REST API listens to incoming JSON requests and delivers them to the Admin API. The Admin API constructs SOAP requests and sends them to the MSSP.

The data tier contains the MSSP databases and thus all the account data to be administrated. The MSSP receives account management operations as MSS SOAP messages.

4.2 Content Server

The content server is an Apache HTTP server that hosts the web interface. It contains the administration HTML pages and JavaScript libraries.

It also acts as a proxy by forwarding traffic between the user's browser and the Admin Server. This allows the browser to get both the content and the services from a single site. Acting as a proxy also allows the Content Server to hide the Admin Server from the Internet.

4.3 Admin Server

The Admin Server consists of the REST API, Admin API and MSSAPI. The REST API provides a JSON message based REST interface for applications and websites. The Admin API handles MReg operation execution. The MSSAPI handles the SOAP communication with the MSSP. Figure 14 shows the components and the communication interfaces of the Admin Server.

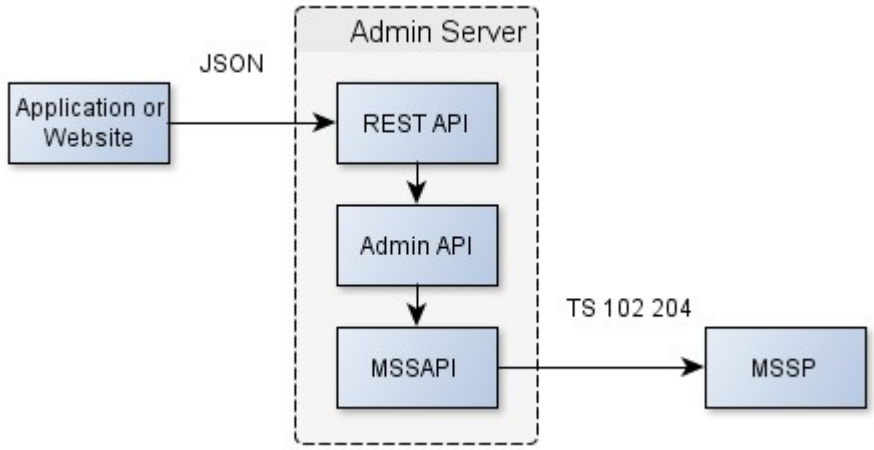

Figure 14: Admin Server Overview

4.3.1 REST API

Kiuru REST Application Interface (Kiuru REST API) is a light-weight REST style interface for Application Providers. The REST API provides easy mobile signature service access. It supports the most common MSS SOAP message formats and operations. The REST API connects to the AE using mutual TLS authentication. [19.]

Kiuru REST API uses JSON data structures which are easy to construct in most software development environments. The following two examples contain a simple JSON signature request and a response for it.

```
{
  "MSS_SignatureReq": {
    "MobileUser": {
      "MSISDN": "+35847001001"
    },
    "MessagingMode": "synch",
    "DataToBeSigned": {
      "MimeType": "text/plain",
      "Encoding": "UTF-8",
      "Data": "Sample text {[hello]}"
    },
    "SignatureProfile": "http://alauda.mobi/digitalSignature"
  }
}
```

Listing 1. REST API Signature Request.

```
{
   "MSS_SignatureResp":{
      "AP_Info":{
         "AP_ID":"http://test1_ap",
         "AP_TransID":"Rheifc4pyT1",
         "Instant":"2014-10-03T13:44:53.782Z"
      },
      "MSSP_Info":{
         "Instant":"2014-10-03T13:44:58.084Z",
         "MSSP_ID":{
            "URI":"http://restapi-he1.methics.fi"
         }
      },
      "MSSP_TransID":"h1pr",
      "MSS_Signature" {
          "Base64Signature":"MIAGCSqGSIb3DQEHAqCAMIACAQExCzAJBgUrDgMCG
          gUAMIAGCSqGSIb3DQEHAaCABBxTYW1wbGUgdGV4dCB7W8O2w6TDpcOWw4TDh
          V19AAAAAKCCA8AwggO8MIICpKADAgECAgFlMA0GCSqGSIb3DQEBCwUAMIGFM
          QswCQYDVQQGEwJGSTERMA8GA1UECAwISGVcmK8pKg8WNjOv+PpQXEgEzUeBK
          pU7YVgOMRVeou0VWv0f0UuvaCPfqAAAAAAAA"
      },
      "MinorVersion":"1",
      "MajorVersion":"1",
      "MobileUser":{
```

```
            "MSISDN":"+35847001001"
         },
         "Status":{
            "StatusCode":{
               "Value":500
            },
            "StatusMessage":"SIGNATURE"
         }
      }
}
```

Listing 2. REST API Signature Response.

The JSON request contains the mobile user's MSISDN and a data block that is being signed. The response contains details about the AP and the MSSP and a signature if the request succeeded.

4.3.2 Admin API

The Admin API is the key component to be implemented for this project. It is a Java library that is used between the REST API and the MSSP.

The Admin API can be used to extend the capabilities of MReg operations by wrapping them into UseCases. The operations within a UseCase are run in an ordered sequence. The UseCases can handle the input and output parameters of the operations in the operation sequence. The parameters are passed between operations with a clipboard.

4.3.3 User Roles

Since the Admin Server will replace many old applications, it will be used by people in different roles. Examples of the possible roles are administrators and end users. The system should provide different views for the different roles, as shown in Figure 15.

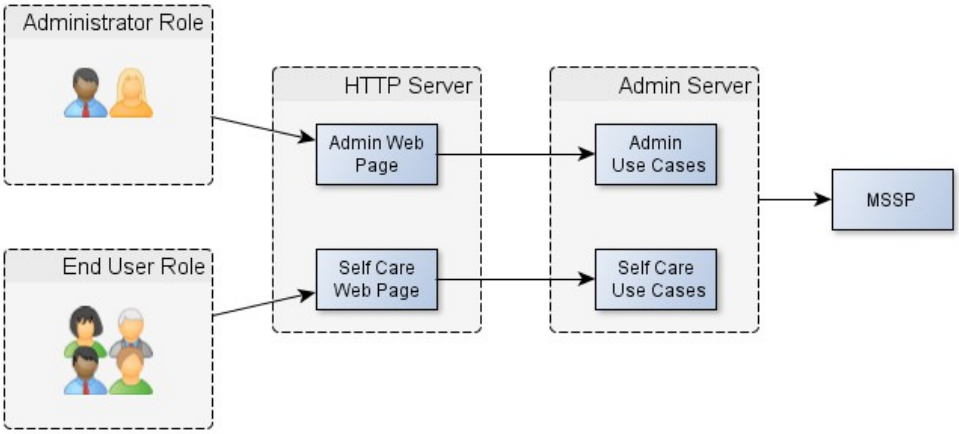

Figure 15: User roles

The list of roles each user is given depends on the user's authentication method. If the user logs in with MSS, they are given the highest possible Level of Assurance (LoA) and all roles. If the user logs in with a less secure authentication method, such as password authentication, they are given a lower LoA and only the roles defined for that LoA.

4.4 MSSP Interface

The MSSP is accessed using MReg and SPML2. To improve the management functionality of the MSSP, MReg implementations of the SPML2 account management operations are created. This improves the pagination functionality of list operations especially, since the SPML2 list and search functions do not support pagination at all.

5 Implementation

This chapter describes the key components of the implementation. The Admin Server implementation consists of Admin API, UseCase definitions and the HTML pages at the Content Server.

The modular design allowed that the implementation could be done piece by piece. The Admin API was the first implemented component. It was initially tested with the MUTK (Mobile User Toolkit) command line tool. After the Admin Server was implemented, the testing moved from MUTK to sending JSON messages directly to the server with tools like Wget. The last implemented component was the Content Server which provides a web interface.

5.1 Admin API

The Admin API is a Java API library that can be used as a part of a client implementation or on a server. It allows combining multiple MSS operations into UseCases. The main components of the Admin API are the Request Dispatcher and the XML configuration files for UseCases and MSpec. (see Figure 16)

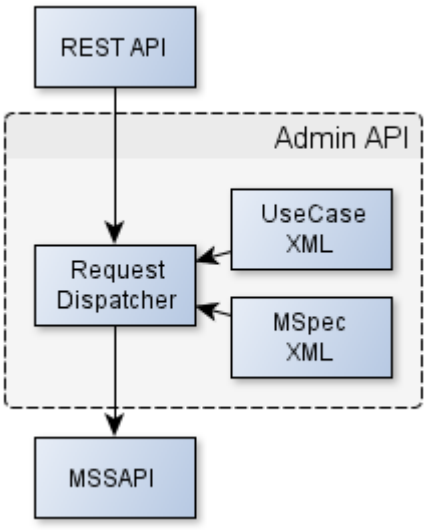

Figure 16: Admin API Structure

Admin API requests contain the identifier and parameters of a UseCase. If a UseCase with that identifier is available, the Request Dispatcher executes the UseCase.

5.1.1 Request Dispatcher

Request Dispatcher is a component in Admin API that handles UseCases and forms one or more MReg or SPML2 requests. It handles the UseCase flow and clipboard. The Request Dispatcher uses MSSAPI for communicating with the MSSP, as seen in Figure 17.

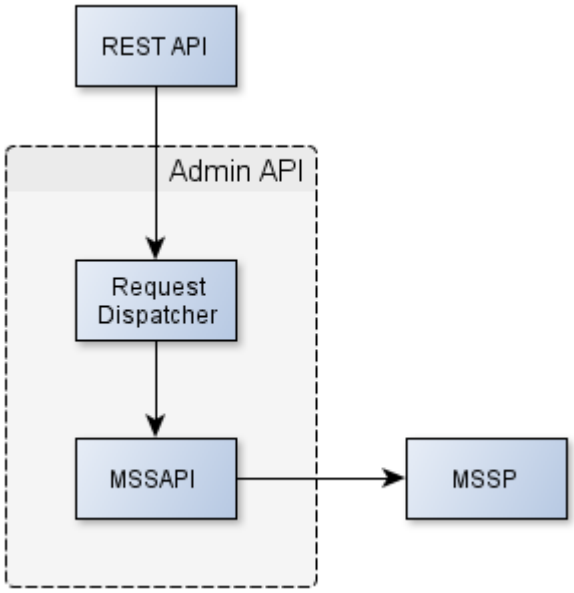

Figure 17: REST API to MSSP

5.1.2 User Roles

The XML configuration files are role specific. Each role has their own folder that can contain multiple UseCase and MSpec files. The following block in Listing 3 shows an example of the roles folder structure with one role called *enduser*.

```
.
└── enduser
    ├── mspec
    │   ├── mspec-operations.xml
    │   └── spml-operations.xml
    └── usecases
        ├── add-ap.xml
        ├── delete-ap.xml
        ├── display-ap.xml
        ├── search-aps.xml
        ├── update-ap.xml
        └── billingreport.xml
```

Listing 3. Role Folder Structure.

When the Admin API is loaded, all configuration files are read for each role. The files are then unmarshalled to Java objects using Castor. The files are reloaded and unmarshalled automatically after they have been modified in the file system.

5.2 UseCases

Most account management operations are simple and only do one task. Because of this, a UseCase system was created. The UseCases support execution of multiple operations in sequence to perform an account management task.

UseCases are XML files that describe a set of MReg or SPLM2 operations to be run in sequence. The operations can have separate input and output parameters. These values can be shared between the operations via a clipboard that is controlled by XML attributes.

A UseCase can be used to describe

- which operations to execute
- in what order to execute the operations
- how to handle operation input and output parameters
- how to pass parameters between operations.

5.2.1 Operations

The operations are described in MReg Specification XML file (MSpec). In order to use an operation in the UseCase file, it must be defined in the used MSpec too. The operations are identified by their name and namespace.

5.2.2 Parameters

An operation can have three types of parameters: InputParam, TargetParam and OutputParam.

Operation input is defined by TargetParams and InputParams. TargetParams are used to deliver special input required by the protocol, for example, MReg target parameters. InputParams are the main method for delivering input for an operation.

OutputParams describe operation results. They are used as output filters. Only operation results that have an OutputParam defined are returned.

5.2.3 Clipboard

The clipboard is a component of the Admin API that passes parameter values between operations. The component can read parameters directly from the UseCase XML file or from the operation output. The clipboard is controlled with the *Copy and Paste* attributes in the UseCase XML.

5.2.4 Example

Creating a new mobile user requires the following series of steps. First the mobile user and SIM card data should be populated to the database. Then the key generation process should be started for the mobile user. After successfully generating a key, the mobile user and the SIM card should be activated. The following listing contains a UseCase for executing the operations.

```
<UseCase Name="Add MobileUser" UseCaseId="add-mobileuser"
         DefaultNameSpace="kiuru">

  <Operation Name="InsertMobileUser">
    <TargetParam Name="MSISDN"             Value="+35847001001" />
    <InputParam  Name="CustomerID"         Value="Acme Inc." />
    <InputParam  Name="CertificateProfile"
                 Value="http://alauda.mobi/digitalSignature" />
    <InputParam  Name="SignatureProfile"
                 Value="http://alauda.mobi/digitalSignature" />
  </Operation>

  <Operation Name="InsertSimCard">
    <TargetParam Name="ICCID"     Value="112233"   Copy="true" />
    <InputParam  Name="MSISDN"    Paste="true" />
    <InputParam  Name="IMSI"      Value="22224444" Copy="true" />
  </Operation>

  <Operation Name="GenerateKey">
    <TargetParam Name="MSISDN" Paste="true" />
  </Operation>

  <Operation Name="ActivateMobileUser">
    <TargetParam Name="MSISDN" Paste="true" />
  </Operation>

  <Operation Name="ActivateSimCard">
    <TargetParam Name="IMSI"      Paste="true" />
    <InputParam  Name="MSISDN"    Paste="true" />
  </Operation>

</UseCase>
```

Listing 4. add-mobileuser UseCase.

In the given example most of the input parameters stay the same. For example all the operations use the same MSISDN. The input parameters are passed in an internal clipboard from one operation to another.

5.3 MSpec

The Admin API reads available MReg operations from an MSpec (MReg Specification) XML file. This file is used to define the operations the target MSSP is capable of executing and which parameters are supported for the operations.

The MSpec files can be used to limit the available operations for the role. An operation has to be defined on the UseCase XML as well as on the MSpec XML to be executed. This way the user can be granted access to modify their own UseCases without giving

them full access to the MSSPs functions. The following listing displays an MSpec operation definition for the *QuerySimCard* operation.

```
<MregOperation Name="QuerySimCard" Type="ProvisioningOperation">
  <Description>
    This operation is used to get SIM-card details.
  </Description>

  <Target Type="SimCard" Choice="true">
    <TargetParam Name="IMSI"   Type="String" Required="false"/>
    <TargetParam Name="MSISDN" Type="String" Required="false"/>
    <TargetParam Name="ICCID"  Type="String" Required="false"/>
  </Target>

  <OutputParam Name="State"        Type="String"/>
  <OutputParam Name="IMSI"         Type="String"/>
  <OutputParam Name="ICCID"        Type="String"/>
  <OutputParam Name="MSISDN"       Type="String"/>
  <OutputParam Name="OperID"       Type="String"/>
  <OutputParam Name="MsgSPIhex"    Type="String"/>
  <OutputParam Name="MsgTARhex"    Type="String"/>
  <OutputParam Name="UsedKeySets"  Type="String"/>
  <OutputParam Name="KeyIndex"     Type="Integer"/>

  <Status StatusCode="100" StatusMessage="OPERATION_OK"
          Success="true"/>
  <Status StatusCode="202" StatusMessage="NO_SUCH_SIMCARD"
          Success="false"/>
  <Status StatusCode="900" StatusMessage="INTERNAL_ERROR"
          Success="false"/>
</MregOperation>
```

Listing 5. QuerySimcard MSpec Operation XML.

5.4 Admin Server

In order to use the Kiuru REST API to run UseCases, new modules are needed. REST API modules can be easily created by implementing an SPI (Service provider interface). These modules are called workers. Each worker handles a single JSON message type.

Two new workers are created for the REST API. RegistrationWorker is made for handling registration requests and AdminWorker for administration requests. Figure 18 displays the relationship between the created workers and the Admin API.

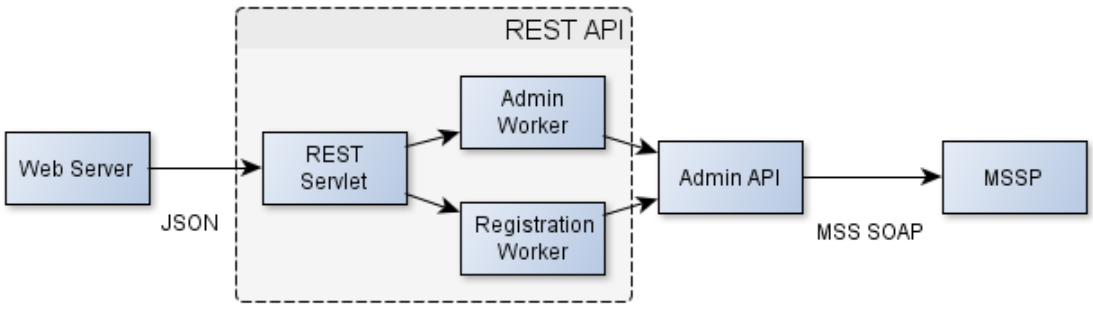

Figure 18: REST API Worker Structure [19.]

5.4.1 Registration Worker

The registration request can pass values directly to the UseCase clipboard by adding InputParam elements. The registration worker prefills the UseCase clipboard and the UseCase determines which parameters from the clipboard to use.

The JSON request must specify the wanted output parameters explicitly with OutputParam elements. Supported input and output parameters are verified and the REST API returns an error message if an unsupported parameter is given or requested. The following listing shows an example of a JSON registration request.

```
{
  "MSS_RegistrationReq": {
    "User": {
      "Role": "enduser"
    },
    "UseCase": {
      "UseCaseId": "add-mobileuser",
      "NameSpace": "kiuru",
      "InputParams": [
        {
          "Name": "MSISDN",
          "Value": "35877887778"
        },
        {
          "Name": "IMSI",
          "Value": "10202030"
        }
      ],
      "OutputParams": [
        {
          "ParamId": "STATE"
        }
      ]
    }
  }
}
```

Listing 6. JSON Registration Request.

5.4.2 Admin Worker

In addition to the worker running UseCases, another worker is needed for UseCase administration. The administration worker is able to add, remove and list supported UseCases. The Java implementation of this worker can be found in Appendix 1. The following two listings show a simple request and a response for listing UseCases.

```
{
  "AdminReq": {
    "User": {
      "Role": "enduser"
    },
    "Operation": {
      "Name": "ListUseCases",
      "NameSpace": "kiuru"
    }
  }
}
```

Listing 7. JSON Administration Request.

```
{
  "AdminResp": {
    "Status": {
      "StatusCode": {
        "Value": "200"
      },
      "StatusMessage": "OK"
    },
    "UseCase": [
      {
        "Name": "InsertMobileUser"
      },
      {
        "Name": "InsertSimCard"
      },
      {
        "Name": "ActivateMobileUser"
      },
      {
        "Name": "ActivateSimCard"
      },
      {
        "Name": "QuerySimCard",
      }
    ]
  }
}
```

Listing 8. JSON Administration Response.

5.5 User Authentication

Authentication for the Admin Server is done with Servlet 3.0 compatible realms. The authentication can be done with either MSS or with traditional password authentication.

MSS authentication is done with MSSAPI. The login form requests the user's MSISDN. AdminRealm then uses MSSAPI to create and a signature request that contains the requested MSISDN. The MSSAPI sends the created signature request to an AE. The AE responds with a signature response. If that response contains a valid signature, the user is authenticated.

Password authentication is done with a simple MemoryRealm implementation. MemoryRealm reads the role definitions and user details from an XML file at the application start. The realm stores the user details in application memory and only reloads them when the server is restarted.

5.6 Admin UI

The Admin UI (User Interface) is an additional component built on top of the Component Server to provide a web UI for the Admin API REST interface. The core idea of the Admin UI is to provide a simple set of operations for a couple of different account types, for example, Mobile Users and Application Providers.

The search bar and the entity type list use standard bootstrap navigation bar components to create a navigation header. (see Figure 19) Selecting an entity type changes the functionality of the search bar to handle that entity type.

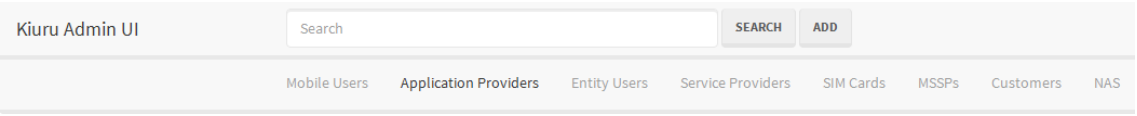

Figure 19: Search Bar. Screenshot [20.]

Search results are shown in a table below the navigation bar. The user can select a result to see more details and to manage the account. Figure 20 shows a search result view containing two Application Providers.

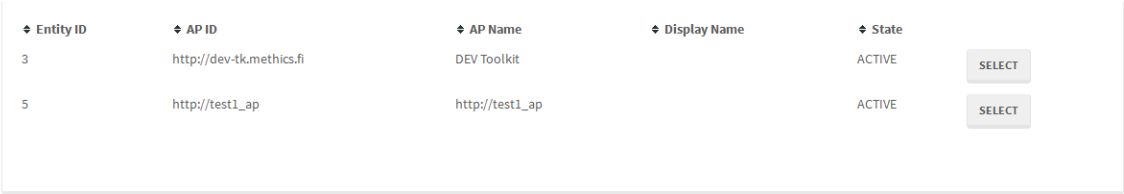

Figure 20: AP Search Results. Screenshot [36.]

6 Testing

6.1 Admin API

For testing the Admin API, a simple extension for the Mobile User Toolkit was created. It can be used to run UseCases in a sequence. The result for each operation is displayed as OK or FAIL depending on the resulting status code.

Simple UseCases were made for all MReg operations to verify that they work with the Request Dispatcher. Listing 9 displays results from a MUTK regression test.

```
add-mobileuser:
  InsertMobileUser:   OK
  InsertSimCard:      OK
  GenerateKey:        OK
  ActivateMobileUser: OK
  ActivateSimCard:    OK
```

Listing 9. MUTK Regression Test Results.

The tests are run on a development environment that has a simulator acting as the wireless end. This way the user interaction with the phone can be avoided.

6.2 REST

During development the Admin Server REST interface was tested with a simple web page that can send any given JSON messages to the server. The test site contains example requests for all UseCase life-cycle management operations. Figure 21 shows the web page with an example request and response.

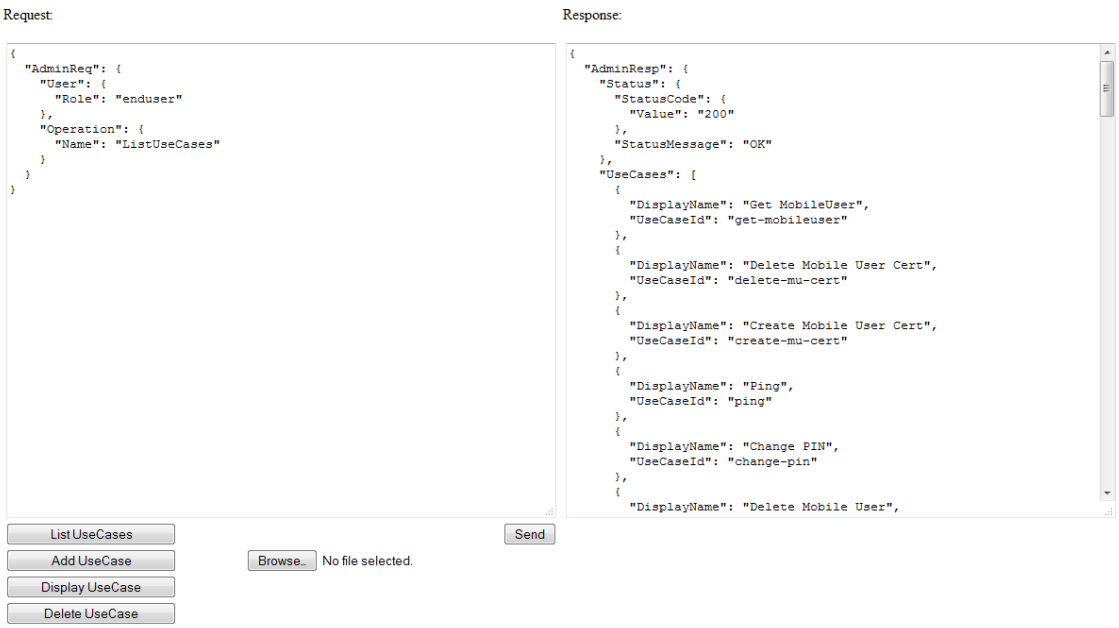

Figure 21: Admin Server REST Test Page. Screenshot [21.]

The Admin REST interface is also regression tested with an open source test framework called ShUnit. ShUnit can be used to run tests from a shell script [Error: Reference source not found.]. The JSON request messages used in the tests are stored as plain text files. The tests use wget to send the requests. Listing 10 shows example output from a ShUnit test.

```
*************** admin_test.sh ****************
4 tests to run:
================================================
    Test 1: List UseCases
------------------------------------------------
Request:  AdminReq_List.txt
Response: AdminResp_List.txt
Assertion Results: ....
================================================
    Test 2: Display UseCase
------------------------------------------------
Request:  AdminReq_Display.txt
Response: AdminResp_Display.txt
Assertion Results: .
================================================
    Test 3: Run 'populate' usecase
------------------------------------------------
Request:  MSS_RegistrationReq_Populate.txt
Response: MSS_RegistrationResp_Populate.txt
Assertion Results: .
================================================
    Test 4: Run 'clean' usecase
------------------------------------------------
```

```
Request:  MSS_RegistrationReq_Clean.txt
Response: MSS_RegistrationResp_Clean.txt
Assertion Results: .
================================================
Results of Suite:
------------------------------------------------

4 tests run.
  4 tests succeeded.
  No tests failed.
```

Listing 10. ShUnit Regression Test Results.

The life-cycle operation testing and registration testing were carried out separately. The tests assume that the MReg operations underneath are working and focus instead on testing the JSON interface. This way both of the test sets stay at around 50 tests each.

7 Conclusions

The goal of the project was to create a secure account management system for the Kiuru MSSP server. The solution was supposed to replace the existing systems with one common framework.

The REST-style approach turned out an excellent extension for SOAP communication. It provides an easy way to integrate new web applications into the MSSP platform. It can be used to replace all Methics' old web based administration applications.

The UseCase role management system fulfills the security requirements of the project. The downside of the system is that every parameter has to be explicitly allowed which can make maintenance hard. The used authorization model recycles other known models which have the same maintenance challenges. From a security perspective the authorization is clearly visible.

Using jQuery and Bootstrap in the web solution is agile compared to the solutions the company's old web applications used. The content server model improves Web user security against cross-site scripting. Additionally the static content server can be connected securely to the Internet and it acts as a firewall.

The biggest part of the project was designing the architecture and validating its components. Since the actual software mostly consists of small modules for existing applications, writing the software was fast. Another big part of the project was learning the technologies and design patterns required for designing a system that can be expanded and maintained.

The biggest part of the software was the Admin API. A full list of its classes and their line counts can be found in Appendix 1. The Admin API consists of 50 classes and around 10,000 lines of code. The extension for the REST API contains only three classes and less than 500 lines of code.

The next obvious step after this project is to modify the rest of the company's old command line tools to use the Admin API. This would also allow expanding the CLI tool

help functionality easily by using the operation, parameter and status descriptions from the MSpec.

References

1 Methics Oy. Mobile Signature Service [online].
URL: http://www.methics.fi/technology/mobile-signature-service/. Accessed 26 November 2014.

2 Methics Oy. Four-corner business model [online].
URL: http://www.methics.fi/technology/mobile-signature-service/four-corner-business-model/. Accessed 25 November 2014

3 ETSI. ETSI TR 102 206 [online]. August 2003.
URL: http://docbox.etsi.org/EC_Files/EC_Files/tr_102206v010103p.pdf.
Accessed 17 November 2014.

4 ETSI. ETSI TS 102 207 [online]. August 2003.
URL: http://docbox.etsi.org/EC_Files/EC_Files/ts_102207v010103p.pdf.
Accessed 2 November 2014.

5 Methics Oy. Kiuru MSSP 5.0 user guide. Helsinki, Finland, 2014.

6 Methics Oy. Kiuru MSSP 5.0 admin users guide. Helsinki, Finland, 2014.

7 Aptana Studio 3 [online].
URL: http://www.aptana.com/products/studio3.html. Accessed 26 November 2014.

8 Apache Ant 1.9.4 manual [online].
URL: http://ant.apache.org/manual/index.html. Accessed 26 November 2014.

9 Oracle. Java Servlet Specification [online].
URL: http://download.oracle.com/otn-pub/jcp/servlet-3.0-fr-eval-oth-JSpec/servlet-3_0-final-spec.pdf. Accessed 2 November 2014.

10 Castor Development team. Castor 1.3.3 – Reference documentation [online].
URL: http://castor.codehaus.org/reference/1.3.3/html/index.html. Accessed 26 November 2014.

11 W3C. SOAP Version 1.2 Part 0: Primer (Second Edition) [online].
URL: http://www.w3.org/TR/2007/REC-soap12-part0-20070427/. Accessed 2 November 2014.

12 ETSI. ETSI TS 102 204 [online]. August 2003.
URL: http://docbox.etsi.org/EC_Files/EC_Files/ts_102204v010104p.pdf.
Accessed 2 November 2014.

13 Fielding R. T. Representational State Transfer (REST) [online]. 2000.
URL: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.
Accessed 2 November 2014.

14 Methics Oy. MReg service management [online].
URL: http://www.methics.fi/technology/mreg-service-management-module/.
Accessed 26 November 2014.

15 Methics Oy. Kiuru MReg Release 5 product fact sheet [online].
URL: http://www.methics.fi/docs/Kiuru_MReg_Product_Factsheet.pdf.
Accessed 18 November 2014.

16 jQuery [online].
URL: http://jquery.com Accessed 18 November 2014.

17 Getting started [online].
URL: http://getbootstrap.com/getting-started. Accessed 18 November 2014.

18 Methics Oy. Kiuru MSSAPI Release 5 product fact sheet [online]. 2013.
URL: http://www.methics.fi/docs/Kiuru_MSSAPI_Product_Factsheet.pdf.
Accessed 18 November 2014.

19 Methics Oy. Kiuru REST API User's Guide. Helsinki, Finland, 2014.

20 Methics Oy. Kiuru Admin UI [computer program]. Helsinki, Finland, 2014.

21 Methics Oy. UseCase life-cycle management [computer program]. Helsinki,
Finland, 2014.

22 ShUnit [online].
URL: http://sourceforge.net/p/shunit/wiki/Home. Accessed 18 November 2014.

Admin API Java Classes

Lines	Java class
658	fi/methics/admin/api/stepper/internal/UseCaseFlow.java
717	fi/methics/admin/api/stepper/internal/StepperSigReqHandler.java
295	fi/methics/admin/api/stepper/internal/StepperMRegHandler.java
753	fi/methics/admin/api/stepper/internal/StepperSPMLHandler.java
309	fi/methics/admin/api/stepper/internal/Clipboard.java
312	fi/methics/admin/api/stepper/internal/UseCaseStep.java
153	fi/methics/admin/api/stepper/RequestDispatcher.java
332	fi/methics/admin/api/stepper/RequestDispatcherImpl.java
150	fi/methics/admin/api/stepper/RequestDispatcherFactory.java
111	fi/methics/admin/api/stepper/Constants.java
255	fi/methics/admin/api/stepper/utils/StepperUtil.java
278	fi/methics/admin/api/stepper/utils/SpecificationsHolder.java
719	fi/methics/admin/api/stepper/utils/UseCaseMSpecMerger.java
180	fi/methics/admin/api/stepper/utils/ParameterTypeChecker.java
289	fi/methics/admin/api/stepper/utils/SpecificationsLoader.java
340	fi/methics/admin/api/stepper/utils/SPMLHelper.java
81	fi/methics/admin/api/stepper/datatype/DispatcherUseCase.java
92	fi/methics/admin/api/stepper/datatype/DispatcherResult.java
45	fi/methics/admin/api/stepper/datatype/DispatcherException.java
221	fi/methics/admin/api/stepper/datatype/DispatcherParameter.java
7	fi/methics/admin/api/stepper/datatype/MergeException.java
25	fi/methics/admin/api/stepper/datatype/DispatcherMspec.java
805	fi/methics/admin/api/MssClient.java
213	fi/methics/admin/api/Conf.java
13	fi/methics/admin/api/InitializationException.java
843	fi/methics/admin/api/SpmlReply.java
94	fi/methics/admin/api/MRegReturnValue.java
895	fi/methics/admin/api/SpmlClient.java
10	fi/methics/admin/api/MRegInterfaceException.java
31	fi/methics/admin/api/MRegOperationResponse.java
63	fi/methics/admin/api/MRegInterface.java
674	fi/methics/admin/api/SpmlInterface.java
168	fi/methics/admin/api/MRegOperationOutput.java
461	fi/methics/admin/api/OperationData.java
10592	Total

REST API JavaScript

```
/**
 * MSS Request message
 *
 * @public
 * @class
 */
function MssRequest () {

    this.request     = null;

    /**
     * @return {string} JSON Request
     */
    this.getRequest = function() {
        return this.request;
    };

    /**
     * @param {string} JSON Request
     */
    this.setRequest = function(request) {
        this.request = request;
    };

    /**
     * Send the request.
     *
     * @param {function} responseCallback
     * @public
     */
    this.send = function(responseCallback) {
        var request = this.request;
        var validate = this.validate;
        var xhr;

        try {
            xhr = new XMLHttpRequest();
            xhr.open("POST", "/rest/service" , true);

            xhr.onreadystatechange = function (event) {
                // do nothing
            };

            xhr.onloadend = function (event) {
                var response = JSON.parse(event.target.responseText);
                responseCallback(response, validate);
            }

            xhr.setRequestHeader('Content-Type',
                                 'application/json; charset=UTF-8');
            xhr.send(JSON.stringify(request));
        } catch(err) {
            console.log("ERROR: " + err);
        }
    };
};
```